下载APP

# 10 | 数据流：如何使用 Vuex 设计你的数据流

2021-11-08 大圣

《玩转Vue 3全家桶》 课程介绍 >

**讲述：大圣**

时长 13:37 大小 12.48M

你好，我是大圣，欢迎进入课程的第 10 讲。

前面的基础入门篇中的几讲，都是针对 Vue 本身的进阶内容。通过这几讲，我们巩固和进阶了 Composition API、组件化和动画等关键知识，Vue 本身的知识点已经掌握得差不多了。那么从这一讲开始，我们进入课程的全家桶实战篇。

领资料

在全家桶实战篇，我们将一同学习 Vue 3 的生态，包括 Vuex、vue-router、Vue Devtools 等生态库，以及实战开发中需要用到的库。这一模块学完，你就能全副武装对复杂的项目开发也会慢慢得心应手。

今天，我先来带你认识一下 Vue 全家桶必备的工具：Vuex，有了这个神兵利器，复杂项目设计也会变得条理更清晰。接下来，让我们先从 Vuex 解决了什么问题说起。

## 前端数据管理

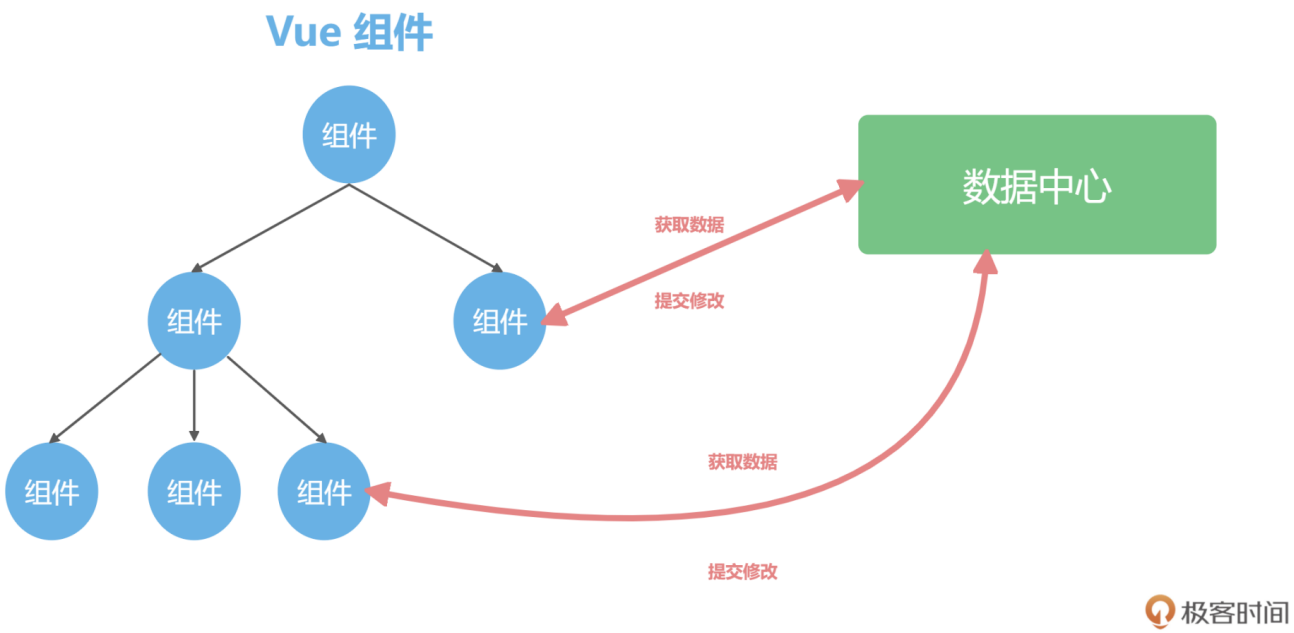
首先，我们需要掌握前端的数据怎么管理，现代 Web 应用都是由三大件构成，分别是：组件、数据和路由。关于组件化开发，在前面的第 8 讲中，已经有详细的讲解了。这一讲我们思考一个这样的场景，就是有一些数据组件之间需要共享的时候，应该如何实现？

解决这个问题的最常见的一种思路就是：专门定义一个全局变量，任何组件需要数据的时候都去这个全局变量中获取。一些通用的数据，比如用户登录信息，以及一个跨层级的组件通信都可以通过这个全局变量很好地实现。在下面的代码中我们使用 _store 这个全局变量存储数据。

```
window._store = {}
```

复制代码

数据存储的结构图大致如下，任何组件内部都可以通过 window._store 获取数据并且修改。




但这样就会产生一个问题，window._store 并不是响应式的，如果在 Vue 项目中直接使用，那么就无法自动更新页面。所以我们需要用 ref 和 reactive 去把数据包裹成响应式数据，并且提供统一的操作方法，这其实就是数据管理框架 Vuex 的雏形了。

## Vuex 是什么

你现在肯定跟小圣有同样的困惑，那就是感觉 Vue 已经够用了，这个 Vuex 又是做什么的？其实，Vuex 存在的意义，就是管理我们项目的数据。

我们是使用组件化机制来搭建整个项目，每个组件内部有自己的数据和模板。但是总有些数据是需要共享的，比如当前登录的用户名、权限等数据，如果都在组件内部传递，会变得非常混乱。

如果把开发的项目比作公司的话，我们项目中的各种数据就非常像办公用品。很多小公司在初创时期不需要管理太多，大家随便拿办公用品就行。但是公司大了之后，就需要一个专门的办公用品申报的流程，对数据做统一地申请和发放，这样才能方便做资产管理。**Vuex 就相当于我们项目中的大管家，集中式存储管理应用的所有组件的状态。**

下面，我们先来上手使用一下 Vuex。我们项目结构中的 src/store 目录，就是专门留给 Vuex 的，在项目的目录下，我们执行下面这个命令，进行 Vuex 的安装工作。

复制代码

```
npm install vuex@next
```

安装完成后，我们在 src/store 中先新建 index.js，在下面的代码中，我们使用 createStore 来创建一个数据存储，我们称之为 store。

store 内部除了数据，还需要一个 mutation 配置去修改数据，你可以把这个 mutation 理解为数据更新的申请单，mutation 内部的函数会把 state 作为参数，我们直接操作 state.count 就可以完成数据的修改。

复制代码

```
import { createStore } from 'vuex'

const store = createStore({
  state () {
    return {
      count: 666
    }
  },
  mutations: {
    add (state) {
      state.count++
    }
```

```
  }
})
```

现在你会发现，我们的代码里，在 Vue 的组件系统之外，多了一个数据源，里面只有一个变量 count，并且有一个方法可以累加这个 count。然后，我们在 Vue 中注册这个数据源，在项目入口文件 src/main.js 中，使用 app.use(store) 进行注册，这样 Vue 和 Vuex 就连接上了。

然后，我们使用 .use 就可以对路由进行注册，使用 .mount 就可以把 Vue 这个应用挂载到页面上，代码如下。

复制代码

```
const app = createApp(App)
app.use(store)
    .use(router)
    .mount('#app')
```

之后，我们在 src/components 文件夹下新建一个 Count.vue 组件，在下面的代码中，template 中的代码我们很熟悉了，就是一个 div 渲染了 count 变量，并且点击的时候触发 add 方法。在 script 中，我们使用 useStore 去获取数据源，初始化值和修改的函数有两个变化：

count 不是使用 ref 直接定义，而是使用计算属性返回了 store.state.count，也就是刚才在 src/store/index.js 中定义的 count。

add 函数是用来修改数据，这里我们不能直接去操作 store.state.count +=1，因为这个数据属于 Vuex 统一管理，所以我们要使用 store.commit（‘add’）去触发 Vuex 中的 mutation 去修改数据。

复制代码

```
<template>
<div @click="add">
    {{count}}
</div>
</template>

<script setup>
import { computed } from 'vue'
```

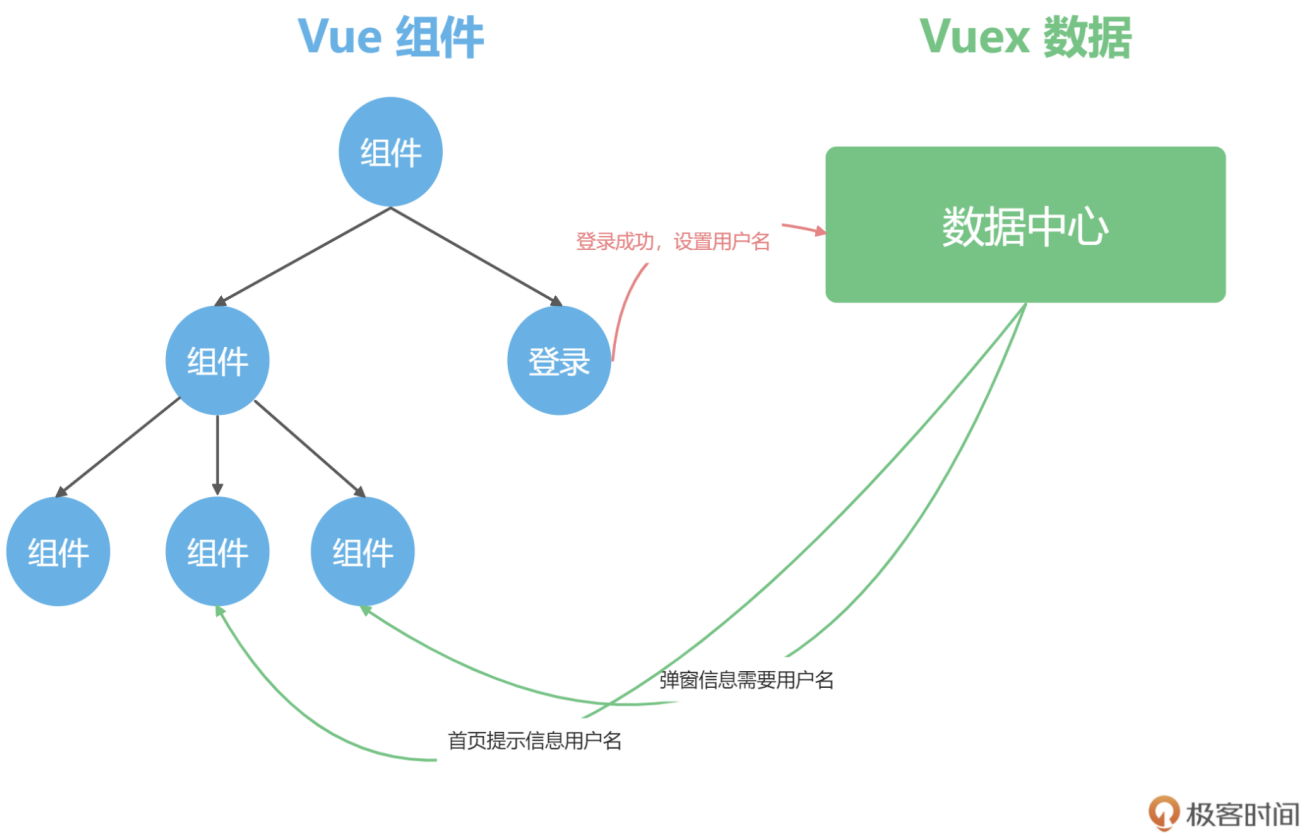
```
import {useStore} from 'vuex'
let store = useStore()
let count = computed(()=>store.state.count)

function add(){
    store.commit('add')
}
</script>
```

在浏览器中打开项目页面，我们就会有一个累加器的效果。相比起来之前用 ref 的方式，真的很简单，这时候小圣就问了我一个问题：什么时候的数据用 Vuex 管理，什么时候数据要放在组件内部使用 ref 管理呢？

答案就是，**对于一个数据，如果只是组件内部使用就是用 ref 管理；如果我们需要跨组件，跨页面共享的时候，我们就需要把数据从 Vue 的组件内部抽离出来，放在 Vuex 中去管理。**

我再结合例子具体说说：比如项目中的登录用户名，页面的右上角需要显示，有些信息弹窗也需要显示。这样的数据就需要放在 Vuex 中统一管理，每当需要抽离这样的数据的时候，我们都需要思考这个数据的初始化和更新逻辑。

就像下图中，项目初始化的时候没有登录状态，我们是在用户登录成功之后，才能获取用户名这个信息，去修改 Vuex 的数据，再通过 Vuex 派发到所有的组件中。




## 手写迷你 Vuex

知道了 Vuex 是什么，接下来我们不妨动手实现一个迷你的 Vuex，这能让你看到 Vuex 的大致原理。

首先，我们需要创建一个变量 store 用来存储数据。下一步就是把这个 store 的数据包转成响应式的数据，并且提供给 Vue 组件使用。在 Vue 中有 provide/inject 这两个函数专门用来做数据共享，provide 注册了数据后，所有的子组件都可以通过 inject 获取数据，这两个函数官方文档介绍得比较详细，我在这里就不过多解释了。

完成刚才的数据转换之后，我们直接进入到 src/store 文件夹下，新建 gvuex.js。下面的代码中，我们使用一个 Store 类来管理数据，类的内部使用 _state 存储数据，使用 mutations 来存储数据修改的函数，注意这里的 state 已经使用 reactive 包裹成响应式数据了。

复制代码

```
import { inject, reactive } from 'vue'

const STORE_KEY = '__store__'
function useStore() {
  return inject(STORE_KEY)
```

```
}
function createStore(options) {
  return new Store(options)
}
class Store {
  constructor(options) {
    this._state = reactive({
      data: options.state()
    })
    this._mutations = options.mutations
  }
}
export { createStore, useStore }
```

上面的代码还暴露了 createStore 去创建 Store 的实例，并且可以在任意组件的 setup 函数内，使用 useStore 去获取 store 的实例。下一步我们回到 src/store/index.js 中，把 vuex 改成 ./gvuex。

下面的代码中，我们使用 createStore 创建了一个 store 实例，并且实例内部使用 state 定义了 count 变量和修改 count 值的 add 函数。

复制代码

```
// import { createStore } from 'vuex'
import { createStore } from './gvuex'
const store = ...
export default store
```

最终我们使用 store 的方式，在项目入口文件 src/main.js 中使用 app.use(store) 注册。为了让 useStore 能正常工作，下面的代码中，我们需要给 store 新增一个 install 方法，这个方法会在 app.use 函数内部执行。我们通过 app.provide 函数注册 store 给全局的组件使用。

复制代码

```
class Store {
  // main.js入口处app.use(store)的时候，会执行这个函数
  install(app) {
    app.provide(STORE_KEY, this)
  }
}
```

下面的代码中，Store 类内部变量 _state 存储响应式数据，读取 state 的时候直接获取响应式数据 _state.data，并且提供了 commit 函数去执行用户配置好的 mutations。

复制代码

```
import { inject, reactive } from 'vue'
const STORE_KEY = '__store__'
function useStore() {
  return inject(STORE_KEY)
}
function createStore(options) {
  return new Store(options)
}
class Store {
  constructor(options) {
    this.$options = options
    this._state = reactive({
      data: options.state
    })
    this._mutations = options.mutations
  }
  get state() {
    return this._state.data
  }
  commit = (type, payload) => {
    const entry = this._mutations[type]
    entry && entry(this.state, payload)
  }
  install(app) {
    app.provide(STORE_KEY, this)
  }
}
export { createStore, useStore }
```

这样在组件内部，我们就可以使用这个迷你的 Vuex 去实现一个累加器了。下面的代码中，我们使用 useStore 获取 store 的实例，并且使用计算属性返回 count，在修改 count 的时候使用 store.commit（‘add’）来修改 count 的值。

复制代码

```
import {useStore} from '../store/gvuex'
let store =useStore()
let count = computed(()=>store.state.count)
function add(){
    store.commit('add')
}

```

恭喜你，这样借助 vue 的插件机制和 reactive 响应式功能，我们只用 30 行代码，就实现了一个最迷你的数据管理工具，也就是一个迷你的 Vuex 实现，下面我们再结合例子，正式介绍一下 Vuex 看一看 Vuex 具体怎么用？

## Vuex 实战

从上面的例子你可以立即看出，Vuex 就是一个公用版本的 ref，提供响应式数据给整个项目使用。现在的功能还比较简单，项目大部分情况都是像之前的清单应用一样，除了简单的数据修改，还会有一些异步任务的触发，这些场景 Vuex 都有专门的处理方式。

在 Vuex 中，你可以使用 getters 配置，来实现 computed 的功能，比如我们想显示累加器数字乘以 2 之后的值，那么我们就需要引入 getters 配置。

下面的代码中，我们实现了计算累加器数字乘以 2 以后的值。我们在 Vuex 中新增了 getters 配置，其实 getters 配置和 Vue 中的 computed 是一样的写法和功能。我们配置了 doubule 函数，用于显示 count 乘以 2 的计算结果。

复制代码

```
import { createStore } from 'vuex'
const store = createStore({
  state () {
    return {
      count: 666
    }
  },
  getters:{
    double(state){
          return state.count*2
      }
  },
  mutations: {
    add (state) {
      state.count++
    }
  }
})

export default store
```

然后，我们可以很方便地在组件中使用 getters，把 double 处理和计算的逻辑交给 Vuex。

复制代码

```
let double = computed(()=>store.getters.double)
```

实际项目开发中，有很多数据我们都是从网络请求中获取到的。在 Vuex 中，mutation 的设计就是用来实现同步地修改数据。如果数据是异步修改的，我们需要一个新的配置 action。现在我们模拟一个异步的场景，就是点击按钮之后的 1 秒，再去做数据的修改。

面对这种异步的修改需求，在 Vuex 中你需要新增 action 的配置，在 action 中你可以做任意的异步处理。这里我们使用 setTimeout 来模拟延时，然后在 action 内部调用 mutation 就可以了。

听起来是不是很绕？不过你不用担心，下面的代码就很清晰地演示了这个过程。

首先，我们在 createStore 的配置中，新增了 actions 配置，这个配置中所有的函数，可以通过解构获得 commit 函数。内部的异步任务完成后，就随时可以调用 commit 来执行 mutations 去更新数据。

复制代码

```
const store = createStore({
  state () {
    return {
      count: 666
    }
  },
  ...
  actions:{
      asyncAdd({commit}){
          setTimeout(()=>{
            commit('add')
          },1000)
      }
  }
})

```

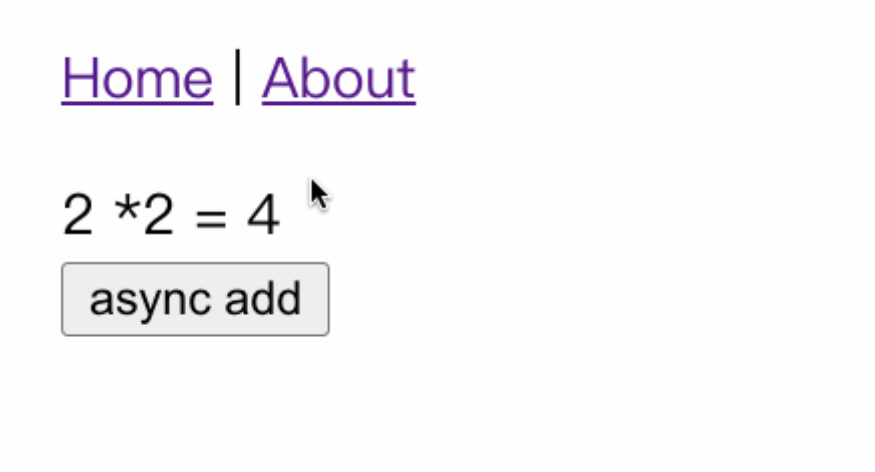
**action 并不是直接修改数据，而是通过 mutations 去修改，这是我提醒你需要注意的。**

actions 的调用方式是使用 store.dispatch，在下面的代码中你可以看到这样的变化效果：页面中新增了一个 asyncAdd 的按钮，点击后会延迟一秒做累加。

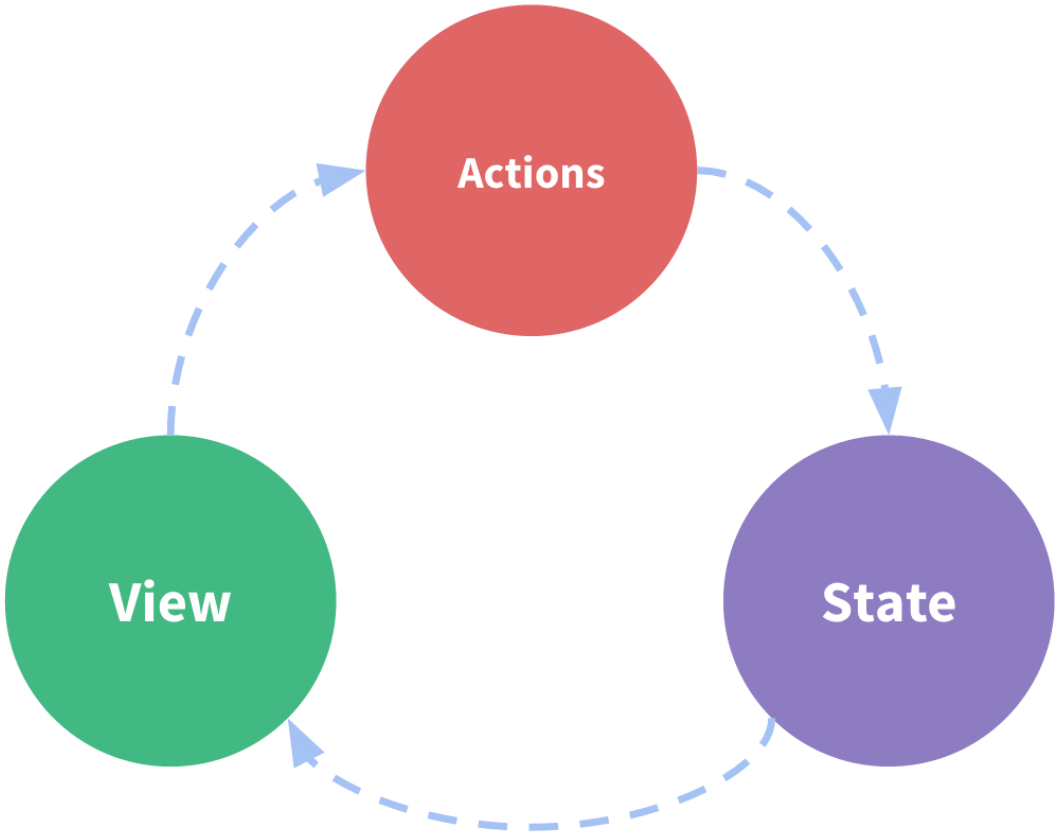
```
function asyncAdd(){
    store.dispatch('asyncAdd')
}
```

复制代码

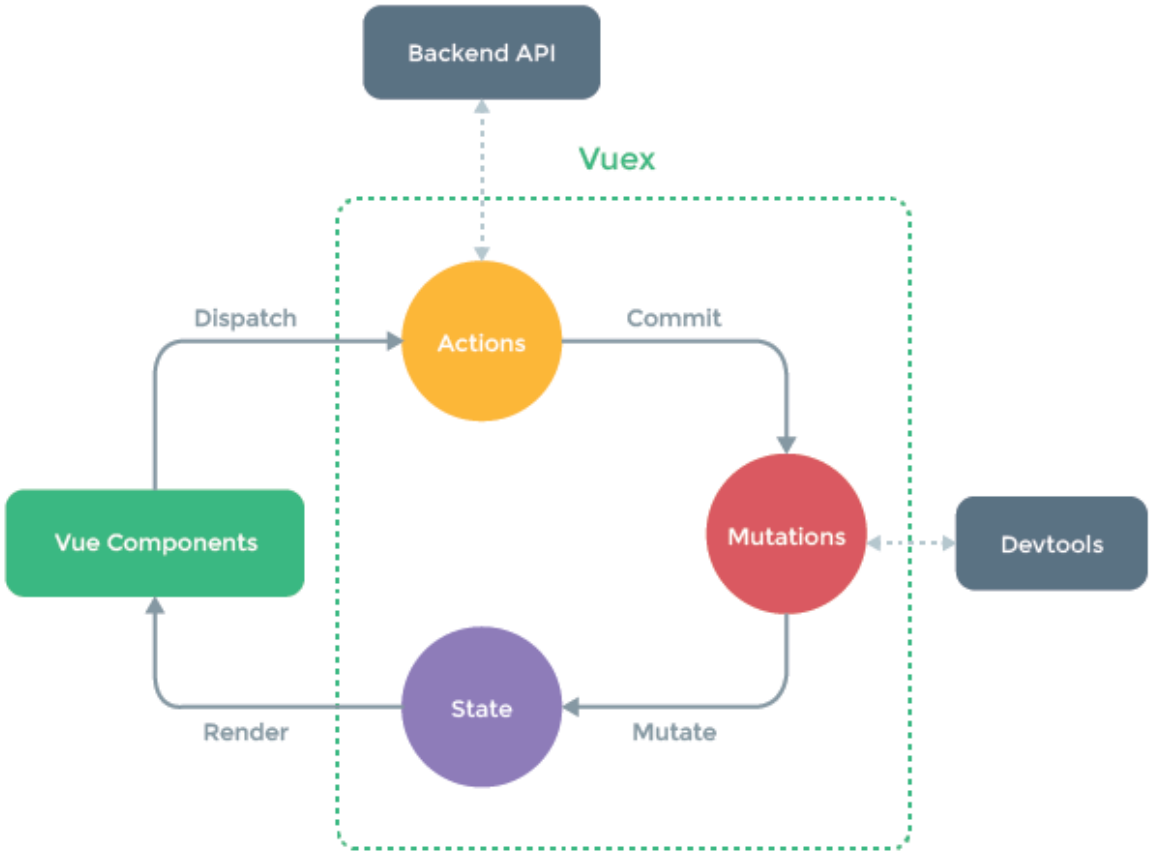
代码执行的效果如下：




Vuex 在整体上的逻辑如下图所示，从宏观来说，Vue 的组件负责渲染页面，组件中用到跨页面的数据，就是用 state 来存储，但是 Vue 不能直接修改 state，而是要通过 actions/mutations 去做数据的修改。




下面这个图也是 Vuex 官方的结构图，很好地拆解了 Vuex 在 Vue 全家桶中的定位，我们项目中也会用 Vuex 来管理所有的跨组件的数据，并且我们也会在 Vuex 内部根据功能模块去做拆分，会把用户、权限等不同模块的组件分开去管理。




由于 Vuex 所有的数据修改都是通过 mutations 来完成的，因而我们可以很方便地监控到数据的动态变化，后面我们可以借助官方的调试工具，非常方便地去调试项目中的数据变化。

回到正在做的这个项目中，有大量的数据交互需求、用户的登录状态、登录的有效期、布局的设置，不同用户还会有不同的菜单权限等。

不过面对眼花缭乱的交互需求，你不能自乱阵脚。总体来说，**我们在决定一个数据是否用 Vuex 来管理的时候，核心就是要思考清楚，这个数据是否有共享给其他页面或者是其他组件的需要**。如果需要，就放置在 Vuex 中管理；如果不需要，就应该放在组件内部使用 ref 或者 reactive 去管理。

## 下一代 Vuex

Vuex 由于在 API 的设计上，对 TypeScript 的类型推导的支持比较复杂，用起来很是痛苦。因为我们的项目一直用的都是 JavaScript，你可能感触并不深，但对于使用

TypeScript 的用户来说，Vuex 的这种问题是很明显的。

为了解决 Vuex 的这个问题，Vuex 的作者最近发布了一个新的作品叫 Pinia，并将其称之为下一代的 Vuex。Pinia 的 API 的设计非常接近 Vuex5 的提案，首先，Pinia 不需要 Vuex 自定义复杂的类型去支持 TypeScript，天生对类型推断就非常友好，并且对 Vue Devtool 的支持也非常好，是一个很有潜力的状态管理框架。

## 总结

今天的学习内容并不难，主要是引入了一个新的框架 Vuex 和数据管理的概念，让我们一起来回顾一下。

首先，我们从前端数据管理概念开始讲起。每个组件内部有自己的数据和模板，那共享的数据怎么科学管理呢？这就需要 Vuex 出马了。

简单来说，Vuex 是一个状态和数据管理的框架，负责管理项目中多个组件和多个页面共享的数据。在开发项目的时候，我们就会把数据分成两个部分，一种数据是在某个组件内部使用，我们使用 ref 或者 reactive 定义即可，另外一种数据需要跨页面共享，就需要使用 Vuex 来进行管理。

之后，我们还讲到了 Vuex 带来了几个新的概念，我们使用 state 定义数据，使用 mutation 定义修改数据的逻辑，并且在组件中使用 commit 去调用 mutations。在此基础之上，还可以用 getters 去实现 Vuex 世界的计算属性，使用 action 来去定义异步任务，并且在内部调用 mutation 去同步数据。

Vuex 的出现，让我们整个项目中的数据流动变得非常自然。数据流向组件，但组件不能直接修改数据，而是要通过 mutation 提出申请，mutation 去修改数据，形成了一个圆环。这种方式对于我们项目的开发、维护和调试都是有很大的帮助。之后，我们一起手写了一个迷你的 Vuex，通过实战巩固前面的学习。

最后，我还简单介绍了一下 Pinia 这个框架，Pinia 算是下一代的 Vuex，感兴趣的同学可以去 Pinia 的官网学习一下。

## 思考题

相信今天的课程结束后，你对 Vuex 会有不一样的了解，那么你的项目里哪些数据要放在 Vuex 中呢？

欢迎在留言区分享你的答案，并和我一起交流讨论，我们下一讲见！

分享给需要的人，Ta订阅后你可得 20 元现金奖励

生成海报并分享

赞 14 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 09 | 动画：Vue 中如何实现动画效果？

下一篇 11 | 路由：新一代 vue-router 带来什么变化

更多课程推荐

## 精选留言 (22)

写留言

置顶

2021-11-08

非常棒的一节，在学习内容的过程中，我也在回顾之前学习关于 Vuex 的知识，在写miniVuex 的实现的代码后，对组件化有了新的认识，简单说下体会。

“组件化”是解决“复杂”问题的重要思想。其实现就是一个个“组件”，即使表现方式不同，核心还是 MVX 的模型。...

展开

作者回复: 精品回复！！赞赞赞

4

**一个小🤚**

2021-11-08

我Vuex4都还没学完，Pinia就出来了，学不完了。
（话说想请教大圣老师，在前端技术发展如此之快的情况下，我们应该如何做取舍呢？）

展开

作者回复: 只学成熟的技术就可以了，不需要过于追热点，Pinia和Vuex也是有很多概念和理念是一脉相承的

10

**乐叶**

2021-11-08

```
constructor(options) {
  this._state = reactive({
    // data: options.state
    data: options.state()
  })...
```

展开

作者回复: 是的是的，第一部分实战代码在这里呀，欢迎遇见bug了提pr呀
https://github.com/shengxinjing/geektime-vue-course

共 4 条评论 >

8

**Geek_4578dc**
2021-11-08

建议大圣老师把每节的代码放出来，这样有利于阅读

展开

作者回复: 第一部分实战代码在这里呀，欢迎遇见bug了提pr呀
https://github.com/shengxinjing/geektime-vue-course

7

**醉月**
2021-11-08

大圣老师Pinia会有涉及吗

展开

作者回复: 最后会有涉及的，毕竟是未来的vuex

3

**南山**
2021-11-08

打卡
1. 登录的用户信息，权限信息，字典信息
2. 全局共享的数据

展开

1

**uncle 邦**
2021-11-08

count 不是使用 ref 直接定义，而是使用计算属性返回了 state.state.count，也就是刚才在 src/store/index.js 中定义的 count。这个 " state.state.count" 是不是要改成 “store.state.count”

作者回复: 感谢提示这个错别字，已经提交给编辑
第一部分实战代码在这里呀，欢迎遇见bug了提pr呀
https://github.com/shengxinjing/geektime-vue-course

1

**葱味黑咖啡**
2021-11-18

第一次使用vuex是在vue2时，用来存储登录的用户数据，但vuex是没有持久化数据的，所以还得配合着localstorage和sessionstorage来用，比较好奇为什么vuex不提供数据持久化的实现？

作者回复: 持久化独立成专门的插件了 vuex-persistedstate

**JIo**
2021-11-17

真心觉得这种设计到多段代码的课程应该出视频课 而不是音频课 各种中英文错综穿插 代码还需要截图在旁边才能听讲 效果真的不如视频来的快。。。

展开

作者回复: 是呀，问题就是这里只有文字和音频 囧 望理解

**百事可乐**
2021-11-16

```
constructor （ ）{

 this.$options = options;
        this._state = reactive({
            data: options.state() // 这里文章这里最后改成 options.state 好像不对吧…
```

展开

作者回复: options.state()是ok的

**轻飘飘过**
2021-11-15

1. 看provide和inject那地方mini-vuex的封装有点懵，说“provide 注册了数据后，所有的子组件都可以通过 inject 获取数据”，但是mini-vuex只是封装了函数并不是组件，后来看官

方文档才知道这个在调用use的时候进行注入了全局组件上了。要是大圣老师再写详细点就好了。
2. 有个疑问，使用app.use(store)进行注入vuex，为啥不封装成vueuse模块，在app.ts中...
展开

**陈豆**
2021-11-14

写个小型的vuex中，参考mutations实现了actions，想请教一下怎么实现getters

**碎竹落棋**
2021-11-13

github源码中，this.getters[name] = computed(() => fn(this.state))，为什么需要用computed去包住呢？
展开

作者回复: 因为getter需要随着状态的变化去更新视图，这就是computed的用武之地了

**瓦力**
2021-11-11

感觉vuex的很大部分逻辑来自于redux，概念基本相同：store、state、action、reducer(mutation)、middleware(commit)，UI与逻辑分离，action触发state改变，并且state是immutable的。

作者回复: 其实都是数据管理的逻辑 通用的 嘿嘿



**Ty丶汤圆圆**
2021-11-10

想问下大圣啊，因为vuex有刷新的问题，所以项目里通常把 vuex 和 Storage 结合在一起做数据持久化，但是用多了就会产生为什么不直接用 Storage 存取数据的疑问。封装后存取数据也一样很方便嘛，还是说仅仅是因为 vuex 有响应式功能呢。希望大圣帮忙解答下，谢谢~
展开

共 3 条评论 >

**无双**
2021-11-09

大圣老师,请问vuex如何处理刷新丢失数据的问题，我自己的处理方式是把Storage封装到vuex中，但感觉不是很好。

展开

**拼搏、进取**
2021-11-08

getters:{ double(state){ return state.count*2 } }, 这个getters如何实现的呢

作者回复: 第一部分的演示代码在这里了，欢迎发现bug提pr
https://github.com/shengxinjing/geektime-vue-course

**Victor**
2021-11-08

vuex是为了解决跨组件和跨页面的大量数据共享的问题，相比较也能够解决此问题的localstorage来说，vuex 管理的数据支持响应式，提供了统一的更新和获取数据的方式。
所以，像用户登录信息，状态信息等都很适合。

展开

**阿阳**
2021-11-08

在实现mini版的vuex时，以下代码this._state = reactive({ data: options.state })，这里是不是应该是options.state()，是一个函数调用。因为在创建store时，state已经写成了一个函数了。

作者回复: mini版本的时候五把state设置为对象了

**1023**
2021-11-08

第一个示例报错，获取不到store是什么原因呢

展开

作者回复: 第一部分实战代码在这里呀，欢迎遇见bug了提pr呀
https://github.com/shengxinjing/geektime-vue-course

共 4 条评论 >